SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October, 1999	Final May 1, 1996 - July 31, 1999

4. TITLE AND SUBTITLE

A Typed and Temporal Object-Oriented Technology

5. FUNDING NUMBERS

DAAH04-96-1-0192

6. AUTHOR(S)

Suad Alagic

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)

Department of Computer Science

Wichita State University

Wichita, KS 67260-0083

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

ARO 35817.16-MA-DPS

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A typed and temporal object-oriented paradigm has been developed. A declarative object-oriented, temporal constraint language MyT, its type system, and a model of persistence have been designed. The results on the associated model theory based on order-sorted algebras and the view of MyT classes as temporal theories have been established. A provably type safe technique called constrained matching has been developed for the integrated typed and temporal object-oriented paradigm. The underlying implementation architecture has been developed based on a persistent extension of the Java Virtual Machine. Specific techniques for handling advanced typing techniques in a persistent Java environment, such as bounded and F-bounded polymorphism, have been developed. Results on the object-oriented flight simulator technology have been established.

14. SUBJECT TERMS	15. NUMBER IF PAGES
Object-oriented technology, declarative programming, temporal logic, type systems, model theory, persistence, Java technology, flight simulation	7
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

REPORT DOCUMENTATION PAGE (SF298)
Continuation Sheet

1 Statement of the Problem

The goal of this project was to develop the basis for a *typed, temporal logic-based, object-oriented technology.* The underlying paradigm includes an *object-oriented type system* with advanced polymorphic features, a suitable *temporal logic basis* and the associated language, and an *execution model.* At least an advanced *prototyping tool* was also planned to be developed in the project. A *flight simulator application* was selected in order to evaluate the benefits of the proposed technology.

A *typed, object-oriented paradigm* underlying this project is equipped with appropriate temporal logic-based, high-level, *constraint specification facilities* suitable for complex systems. Contrary to the usual situation in strongly typed object-oriented programming languages, programming in the proposed environment is *declarative.* The required logic basis is required to have *well-defined semantics*, but at the same time an object-oriented execution model.

Executable specifications were intended to be used in two ways. Their (almost) direct executability leads to a prototyping tool for strongly typed object-oriented systems. When subject to a suitable implementation technique, those specifications become a basis for object-oriented query and other database languages. The project was also intended to explore a prototype implementation based on a lower-level support of an existing, experimental, *persistent object manager.*

2 List of Publications

2.1 Published papers

S. Alagić, Temporal object-oriented programming, *Object-Oriented Systems, 6*, pp. 1-42, 1999.

S. Alagić, Constrained matching is type safe, Proceedings of the 6th International Workshop on Database Programming Languages (DBPL), 1997, *Lecture Notes in Computer Science 1369*, pp. 78- 96, 1998.

S. Alagić, J. Solorzano and D. Gitchell: Orthogonal to the Java imperative, *Lecture Notes in Computer Science 1445*, pp. 212-233, 1998.

J. Solorzano and S. Alagić, Parametric polymorphism for Java: A reflective solution, Proceedings of OOPSLA '98, pp. 216-225, ACM, 1998.

S. Alagić, Flight simulator database: Object-oriented design and implementation, In: A. Chaudhri and M. Loomis, *Object Databases in Practice*, pp. 79-94, Prentice-Hall, 1998.

S. Alagić, The ODMG object model: does it make sense?, Proceedings of the OOPSLA '97 Conference, pp. 253-270, ACM, 1997.

S. Alagić and M. Alagić, Order-sorted model theory for temporal executable specifications, *Theoretical Computer Science*, 179, pp. 273-299, 1997.

S. Alagić, A temporal constraint system for object-oriented databases, Proceedings of the Workshop on Constraints and Databases, *Lecture Notes in Computer Science*, Springer-Verlag, 1191, pp. 298-218, 1997.

S. Alagić, G. Nagati, J. Hutchinson and D. Ellis, Object-oriented flight simulator technology, Collection of Technical Papers, AIAA Conference, pp. 360-368, 1996.

2.2 Accepted papers

S. Alagić, Type checking OQL queries in the ODMG type systems, *Transactions on Database Systems*, 1999, to appear.

2.3 Papers under review

S. Alagić, Semantics of temporal classes, under review for *Information and Computation*, submitted 1997.

3 Scientific Personnel

3.1 Principal investigator

Dr. Suad Alagić, Professor

Senior Faculty Fellow, National Institute for Aviation Research

Academic Reviewer Member, ODMG (Object Database Management Group)

Member, Java Data Objects, Expert Group, Sun.

Member, Round Table Group.

NSF Research Grant: A Family of the ODMG Object Models, 1998 - 20001.

3.2 Graduate students

- Svetlana Kouznetsova, Ph.D. student, degree expected to be completed in Spring 2000.
- Jose Solorzano, M.S. degree completed.
- David Gitchell, M.S. project still to be completed
- Tuong Nguyen, M.S. project in progress.

- Chor-Hiong Law, M.S. degree completed
- Mei Chu Kennis Lai, M.S. degree completed

4 Summary of the Results

4.1 The integrated typed and temporal paradigm

A strongly typed, declarative, temporal logic based paradigm has been developed. A temporal, persistent constraint language MyT based on this paradigm has been designed. Its prototype implementation has been carried out. The paradigm, the overall declarative programming environment, and the implementation model are presented in the paper *Temporal object-oriented programming.*

Temporally constrained matching in a persistent and declarative object-oriented system MyT is introduced as a semantic alternative to the existing approaches to the covariance/contravariance problem. While the existing object-oriented type systems are based on subtyping, F-bounded polymorphism and matching, this technique is based entirely on inheritance, which is identified with matching. The type of matching used in this technique relies on the temporal constraint system.

We proved that this constrained matching guarantees type safe substitutability even in situations where matching alone would not. This is possible only because the underlying formal system of MyT is semantically much richer than the paradigms of type systems. Its temporal constraint system can capture subtleties that go far beyond the level of expressiveness of object-oriented type systems. These results are published in our DBPL paper *Constrained matching is type safe.*

4.2 Persistence and database issues

In developing the type system and the model of persistence for the technology developed in this project, a substantial effort has been devoted to studying other related and highly visible technologies. Among those, the technology of ODMG Standard was particularly carefully scrutinized since it represents the current thinking of the major industrial representatives of object-oriented database technology.

In our OOPSLA'97 paper *The ODMG Object Model: Does it make sense?*, the ODMG Object Model was shown to have a number of problems. A major confusion is caused by the intended type of polymorphism and the way it is expressed in the Model. Dynamic type checking is required even in situations when static type checking is possible. There are situations in which there is no way that type checking can determine whether a particular construct is type correct or not.

The model of persistence in the ODMG Standard is not orthogonal, which has undesirable pragmatic consequences on complex objects. The discrepancies between the ODMG Object Model and the particular language bindings of the ODMG Standard are non-trivial. Our OOPSLA'97 paper presents solutions to some of these problems together with the associated

formal system. Without such a formal system the recommended ODMG bindings are open to a wide range of different, and sometimes confusing interpretations.

Surprising and disturbing negative results are proved about the ability to type check queries in the type systems of the ODMG Standard. The first of these negative results is that it is not possible to type check OQL queries in the type system underlying the ODMG Object Model and its definition language ODL. The second negative result is that OQL queries cannot be type checked in the type system of the Java binding of the ODMG Standard either. An expected positive result is that type checking of OQL queries presents no problem for the type system of the C++ binding of the ODMG Standard.

Different options are outlined for fixing the ODMG Object Model and for extending the type systems of its language bindings in order to make type checking of OQL queries possible. Results that clarify when static versus dynamic type checking of OQL queries is possible are also established. A type system that is strictly more powerful than any of the type systems of the ODMG Standard is required in order to type properly ordered collections and indices.

The above results are presented in our paper *Type checking OQL queries in the ODMG type systems.* This paper and the ODMG related research is the focus of the NSF Research Grant *A Family of ODMG Object Models.*

4.3 Typed and temporal model theory

A model theory of a typed, declarative, temporal object-oriented language system, has been developed with results on classes as temporal theories. These results are tied precisely to the types of temporal constraints available in MyT. The model theory is based on temporal order-sorted algebras with predicates. A variety of orderings are explored in order to represent various types of inheritance, as well as the subtyping. Temporal classes are viewed as temporal theories and some inheritance relationships as morphisms of temporal theories.

A model of a temporal class is a temporal order-sorted structure with predicates which satisfies a set of temporal constraints specified in that class. Morphisms of those models are naturally required to preserve type coercions. A distinguished model of a temporal theory is constructed as a colimit of a suitably defined functor. This colimit construction reflects the temporal nature of the paradigm and generalizes the classical initial algebra semantics. In contradistinction to major difficulties in developing a model theory for full-fledged, typed procedural object-oriented languages, these results show that such a task becomes possible for a suitably defined declarative object-oriented language.

Some of the above model theoretic results were published in our *Theoretical Computer Science* paper *Order-sorted model theory for temporal executable specifications*, but the more recent results on classes as temporal theories appear only in the paper *Semantics of temporal classes*, which has been submitted to *Information and Computation.* This paper is still under review.

4.4 Type system

The interplay of parametric polymorphism and type-safe reflection was carefully investigated. These results were published in the OOPSLA '98 paper *Parametric polymorphism for Java: A reflective solution.*

These results apply to the most serious open problem in the Java programming language: lack of parametric polymorphism. Amidst a variety of existing proposals the results reported in the above paper have a distinguishing characteristic: they are compatible with Java Core Reflection.

A number of inadequacies of existing implementation techniques for extending Java^{TM} with parametric polymorphism are revealed. Implementation techniques that address these concerns are developed. In languages that support run-time reflection, an adequate implementation of parametric, bounded and F-bounded polymorphism is shown to require (reflective) run-time support. In Java, extensions to the core classes are needed. This is in spite of the fact that parametric polymorphism is intended to be managed *statically.*

The results of these investigations have implications on the underlying implementation architecture for the temporal object-oriented constraint language MyT.

4.5 System implementation architecture

Java Virtual Machine extended with persistence capabilities was chosen as the underlying system support. The compiler of the typed, object-oriented, temporal language MyT has been largely implemented in such a way that it generates byte code of the Java Virtual Machine. A distinguished orthogonal model of persistence of this language has been implemented on top of PJama, a persistent supporting extension of Java.

MyT makes it possible to express preconditions, postconditions and class invariants. This addresses the problem of the lack of assertions in Java. MyT has been integrated into the Java programming environment by allowing references to Java classes. On the other hand, Java classes can access persistent objects and their classes, which are created by MyT.

MyT features an orthogonal model of persistence and transitive persistence (persistence by reachability). Unlike all other approaches, persistence capabilities in this model are associated with the root (top) class. This way all classes are persistence capable. The model is naturally based on reachability. In addition, this model of persistence features hierarchical name space management, extending the existing Java mechanism based on packages. Such complex name space management is lacking in persistent supporting Java extensions, such as PJama.

The constraint language supports parametric polymorphism, thus overcoming a serious drawback of Java in database applications: static typing of collections. The form of parametric polymorphism is F-bounded, so that it allows proper typing of ordered collections and indices. A specific implementation technique for implementing F-bounded polymorphism in a persistent Java environment has been developed.

Unlike the existing Java preprocessor systems, this system is a real compiler that generates

Java byte code. In addition, a real novelty of this compiler is that it generates methods from declarative constraints. This in particular includes proper treatment of preconditions, postconditions and class invariants in the generated code. A class in the declarative temporal constraint language is translated into a Java *Class* file.

The constraint language naturally supports queries. Queries refer to persistent collections. They can be statically type checked, which would not be possible in the type system underlying Java.

4.6 Flight simulator application

There have been two papers reporting the developments related to the main targeted application area of this project: *Object-oriented flight simulator technology* and *Flight simulator database: object-oriented design and implementation.*

The fundamentals of a new, object-oriented software technology, for the design and implementation of flight simulators are presented in these papers. The technology offers a powerful paradigm, and a methodology for complex structural and behavioral modeling. The underlying complex software required to support a sophisticated flight simulator is organized into a hierarchy of classes. Inheritance is used as a modeling technique, but at the same time, it allows software reusability. The main contribution of this approach is that it demonstrates that the object-oriented technology makes it possible to design and implement a generic flight simulator as a collection of general classes. This class library can then be used in such a way that specific flight simulators can be derived by inheritance from the generic one. The advantages of this approach are in major reductions in the required effort and cost. Yet another contribution is to demonstrate that the object-oriented database technology is a superior generic database support for the design, implementation and use of flight simulators.